MINISTRY OF SCIENCE AND EDUCATION
NATIONAL TECHNICAL UNIVERSITY
“KHARKIV POLYTECHNIC INSTITUTE”
DEPARTMENT OF SOFTWARE ENGINEERING AND MANAGEMENT
INFORMATION TECHNOLOGIES

METHODICAL RECOMMENDATION TO

“BASICS OF SOFTWARE ENGINEERING
LABORATORY PRACTICE
PART 3”

For students of specialties
121 “Software Engineering”,
122 “Computer Science and Intellectual Systems”

Kharkiv
NTU “KhPI”
2019

Methodical recommendation to “Basics of software engineering. Laboratory practice. Part 1” for students of Technical science / authors Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A. – Kharkiv : NTU “KhPI”. – 17 p.

Authors Melnyk K.V.,
Borisova N.V.,
Lutenko I.V.,
Ershova S.I.,
Smolin P.A.,
Grinchenko M.A.

Reviewer Shmatko A.V.

Department of Software Engineering and Management Information Technologies

CONTENT

INTRODUCTION

Software engineering is an engineering discipline that is concerned with all aspects of software production [1].

Software engineering can be divided into sub-disciplines [2]. Some of them are:

- Software engineering management: The application of management activities – planning, coordinating, measuring, monitoring, controlling, and reporting – to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
- Requirements engineering: The elicitation, analysis, specification, and validation of requirements for software.
- Software design: The process of defining the architecture, components, interfaces, and other characteristics of a system or component.
- Software construction: The detailed creation of working, meaningful software through a combination of programming, verification, unit testing, integration testing, and debugging.
- Software testing: An empirical, technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.
- Software maintenance: The totality of activities required to provide cost-effective support to software.
- Software quality.

The outcome of software engineering is an efficient and reliable software product. Software product is a computer programs with all associated documentation and configuration data that is required to make these programs operate correctly [1]. Essential attributes of good software product: maintainability, dependability and security, efficiency, acceptability.

Authors: Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A.
Department of SEMIT, NTU «KhPI»
Kharkiv, 2019

A software process is a sequence of activities that leads to the production of a software product. There are four fundamental activities that are common to all software processes. These activities are:

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.

2. Software development, where the software is designed and programmed.

3. Software validation, where the software is checked to ensure that it is what the customer requires.

4. Software evolution, where the software is modified to reflect changing customer and market requirements.

The methodical recommendation is about modelling (or designing) and creating a software that will help to calculate some system of expressions with unknown variables. The values of variables can be obtained from different sources: from file or from keyboard.

Authors: Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A.
Department of SEMIT, NTU «KhPI»
Kharkiv, 2019

LAB № 5

TESTING OF CODE

Goal: Learning basic principles of testing C++ code

Tasks:

1. Study principles of using functions in C++.
2. Study Exception Handling in C++.
3. Modify the code from lab 2 according to 1 and 2 tasks.
4. Implement unit testing for developed program.
5. Make all necessary actions on xp-dev.com. Show the iteration where you made updating information.
6. Prepare the report of the work

Progress of the lab.

1. Study principles of using functions in C++.

You should modify the code from the previous lab by using functions in C++.

A function is a group of statements that is given a name, and which can be called from some point of the program. Functions allow to structure programs in segments of code to perform individual tasks. The main advantage is a function can actually be called multiple times within a program.

Depending on whether a function is predefined or created by programmer; there are two types of function:

1. **Library Function**. Programmer can use library function by invoking function directly; they don't need to write it themselves.
2. **User-defined Function.** A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).

Authors: Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A.
Department of SEMIT, NTU «KhPI»
Kharkiv, 2019

The most common syntax to define a function is:

```
type name ( parameter1, parameter2, ...) { statements }
```

where:

- `type` is the type of the value returned by the function; if no value is returned to the calling function then, `void` should be used;

- `name` is the identifier by which the function can be called;

- `parameters` (as many as needed): The purpose of parameters is to allow passing arguments to the function from the location where it is called from;

- `statements` is the function's body.

2. Study Exception Handling in C++.

An exception is an error condition that prevents the program from continuing along its regular execution path. Certain operations, including object creation, file input/output, and function calls made from other modules, are all potential sources of exceptions even when your program is running correctly. Robust code anticipates and handles exceptions. The C++ language provides built-in support for throwing and catching exceptions. To implement exception handling in C++, you use `try`, `throw`, and `catch` expressions.

```
try
{
    // ...

    throw 1;

}

catch( ... )

{

    // Handle exception here.

}
```

Authors: Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A.
Department of SEMIT, NTU «KhPI»
Kharkiv, 2019

Use a `try` block to enclose one or more statements that might throw an exception. A `throw` expression signals that an exceptional condition – often, an error – has occurred in a `try` block. The `throw` statement behaves much like `return`. You can use an object of any type as the operand of a `throw` expression. Typically, this object is used to communicate information about the error. To handle exceptions that may be thrown, implement one or more `catch` blocks immediately following a `try` block. Each `catch` block specifies the type of exception it can handle.

3. Modify the code from lab 2 according to 1 and 2 tasks.

The function “Calculate” shows how to calculate value on definite range of expression (2.1) from lab 2 (Figure 5.1).

This expression has one specific condition: the value of n should be greater than 4. This constraint was developed in function “checkValidParams” (Figure 5.1).

If user enter incorrect value for `n` and `x`, for example, character or double value, then program should generate exception. These cases are reflected in the function “checkValidInput”.

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
using namespace std;

void checkValidInput()
{
    if (cin.fail())
    {
        throw "Incorrect input";
    }
}

void checkValidParams(int n)
{
    if (n < 4)
    {
        throw "Input correct data";
    }
}

int calculate(int n, int x)
{
    int y = 0;

    if (x < 7)
    {
        for (int i = 2; i <= n - 2; i++)
        {
            y += pow((x - i), 2);
        }
    }
    else
    {
        y = x + 7;
    }

    return y;
}
```

Figure 5.1 – Part of the code

The main function of developed code is on the Figure 5.2.

```
41  int main()
42  {
43      int x, n;
44
45      try
46      {
47          cout << "Input n>=4, n=";
48          cin >> n;
49          checkValidInput();
50          checkValidParams(n);
51
52          cout << "Input x=";
53          cin >> x;
54          checkValidInput();
55
56          cout << "x= " << x << ";   " << "y= " << calculate(n, x) << endl;
57      }
58      catch (const char* ex)
59      {
60          cout << ex << endl;
61          return -1;
62      }
63      catch (...)
64      {
65          cout << "Unknown error" << endl;
66          return -2;
67      }
68  }
```

Figure 5.2 – The “Main” function

4. Implement unit testing for developed program

By this moment you should have been developed a Win32 Console Application with all necessary functions for solving the equation. To test these functions, you have to create unit tests for each function separately. Sometimes it is necessary to create several unit tests for one function. It is done with the aim to fully test the behavior of each function with all possible input parameters.

To create unit tests for the existing project, select File → Add → New Project (Figure 5.3). Select Native Unit Test Project (Figure 5.4).

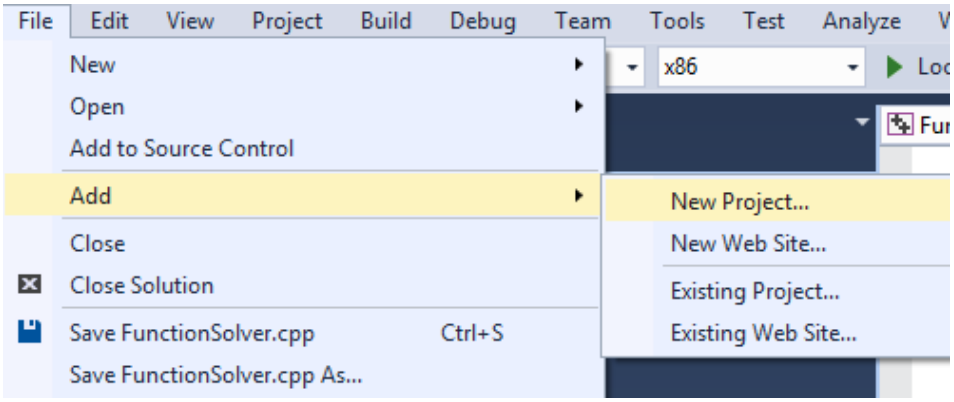


Figure 5.3 – The process of creating of unit test project

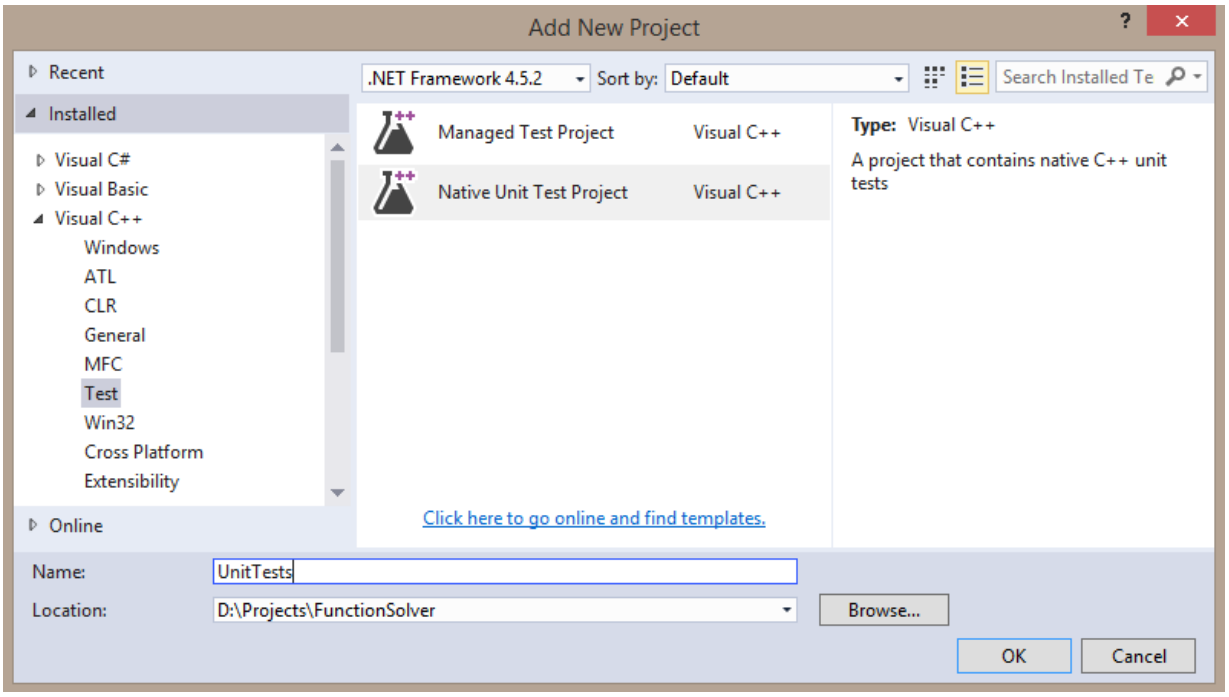


Figure 5.4 – The process of creating of unit test project

You can see unit test project in the Solution Explorer (Figure 5.5).

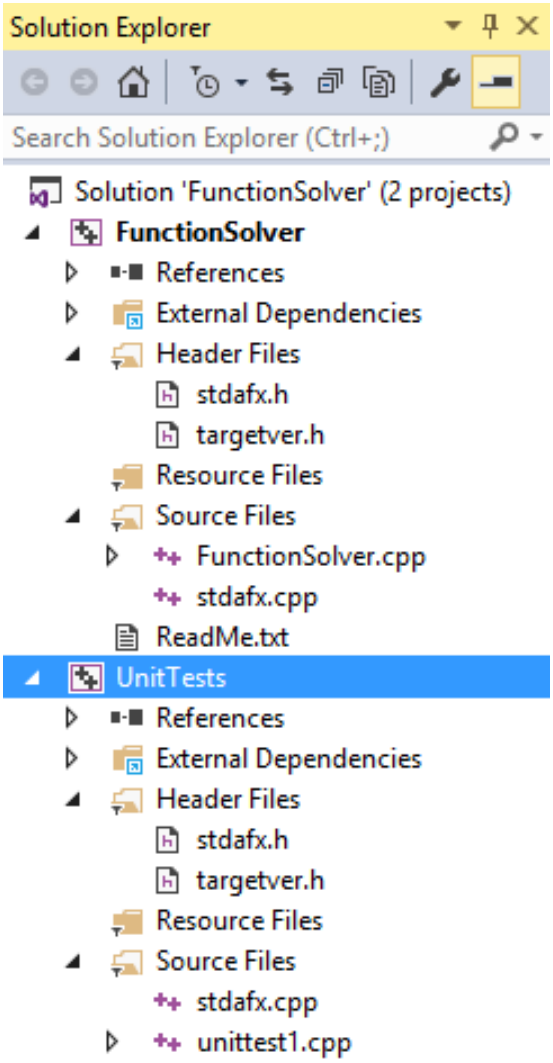


Figure 5.5 – Solution Explorer

Open the file unittest1.cpp. Unit tested will be placed in it. Right click on your source cpp file and choose “Copy Path” menu item (Figure 5.6). Paste this path to the file that will contain your unit tests (Figure 5.7).

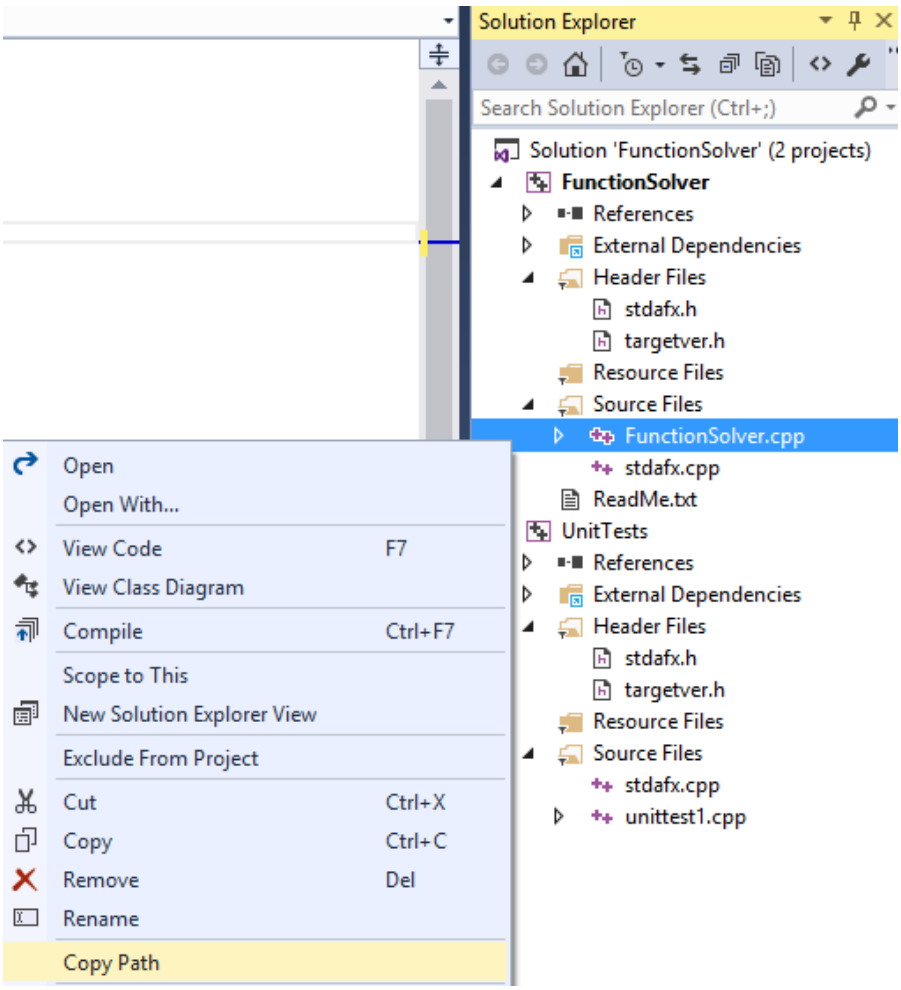


Figure 5.6 – Using source cpp file in test project (1)

```
1   #include "stdafx.h"
2   #include "CppUnitTest.h"
3   #include "D:\Projects\FunctionSolver\FunctionSolver\FunctionSolver.cpp"
4
5   using namespace Microsoft::VisualStudio::CppUnitTestFramework;
6
7   namespace UnitTests
8   {
9
10  }
```

Figure 5.7 – Using source cpp file in test project (2)

Create tests for your code. Each test is defined by using `TEST_METHOD(YourTestName){...}`.

You do not have to write a conventional function signature. The signature is created by the macro TEST_METHOD. The macro generates an instance function that returns void. It also generates a static function that returns information about the test method. This information allows the test explorer to find the method.

Test methods are grouped into classes by using `TEST_CLASS(YourClassName){...}`.

When the tests are run, an instance of each test class is created. Create the test methods names in the following manner:

`testedFunction_functionArguments_expectedResult`.

For testing purposes use several different methods on your choice from the Assert class. You can read about this class and its methods on MSDN https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx.

The example of tests is shown on Figures 5.8-5.9.

```
#include "stdafx.h"
#include "CppUnitTest.h"
#include "D:\Projects\FunctionSolver\FunctionSolver\FunctionSolver.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTests
{
    TEST_CLASS(calculate_Tests)
    {
    public:
        TEST_METHOD(calculate_get4and7_14returned)
        {
            int n = 4;
            int x = 7;
            int expected = 14;

            int actual = calculate(n, x);

            Assert::AreEqual(expected, actual);
        }

    public:
        TEST_METHOD(calculate_get15and6_170returned)
        {
            int n = 15;
            int x = 6;
            int expected = 170;

            int actual = calculate(n, x);

            Assert::AreEqual(expected, actual);
        }
    };
```

Figure 5.8 – Unit tests for Calculate function

```
36      TEST_CLASS(checkValidParams_Tests)
37      {
38      public:
39          TEST_METHOD(checkValidParams_get10_exceptionNotThrown)
40          {
41              int n = 10;
42
43              try
44              {
45                  checkValidParams(n);
46                  Assert::IsTrue(true);
47              }
48              catch (...)
49              {
50                  Assert::Fail();
51              }
52          }
53
54      public:
55          TEST_METHOD(checkValidParams_get3_exceptionThrown)
56          {
57              int n = 3;
58
59              try
60              {
61                  checkValidParams(n);
62                  Assert::Fail();
63              }
64              catch (...)
65              {
66                  Assert::IsTrue(true);
67              }
68          }
69      };
70  }
```

Figure 5.9 – Unit tests for checkValidParams function

When test methods are developed, you can execute them. To do that, you have to build solution (Build → Build Solution / Ctrl + Shift + B), open Test Explorer if it wasn't opened automatically (Test → Windows → Test Explorer) and run all tests. If everything go smoothly, you will see that all tests passed in the summary sub-window (Figure 5.10).

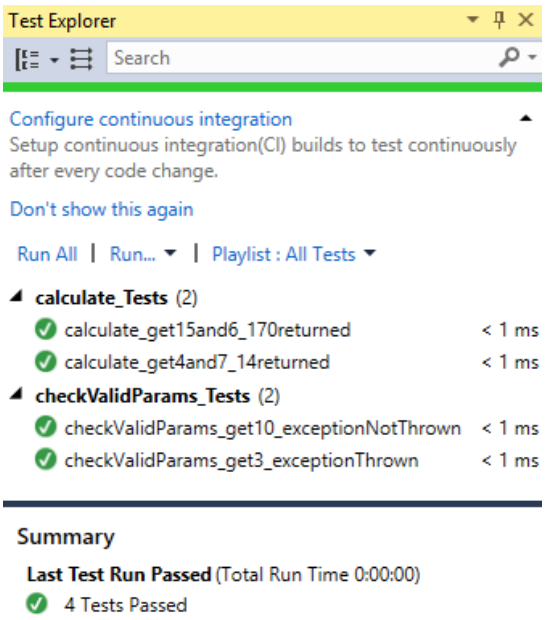


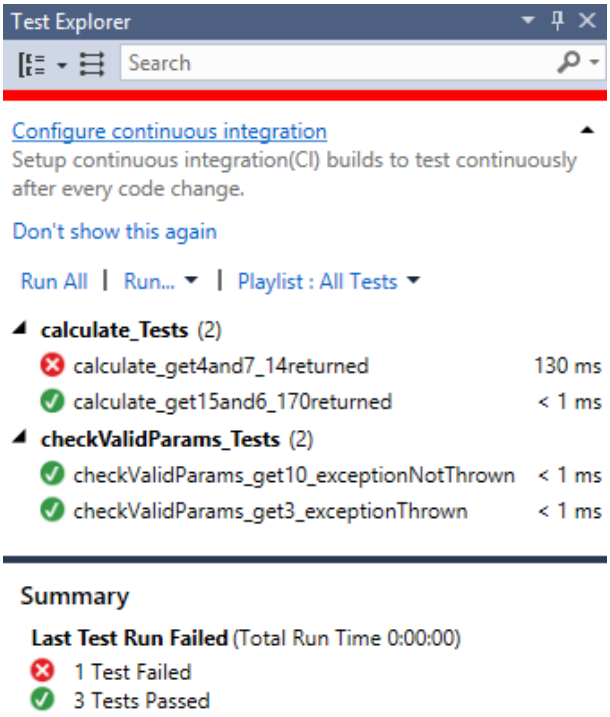
Figure 5.10 – Successful unit tests

Also you should make some changes in unit tests to get them unpassed. For example, change the expected result to the wrong value and run the test again (Figure 5.11 a). In the Test Explorer you will see which tests weren't passed (Figure 5.11 b).

```
TEST_METHOD(calculate_get4and7_14returned)
{
    int n = 4;
    int x = 7;
    int expected = 100000000;

    int actual = calculate(n, x);

    Assert::AreEqual(expected, actual);
}
```



(a) (b)

Figure 5.11 – Unit test with error

5. Make all necessary actions on [xp-dev.com](xp-dev.com). Show the iteration where you made updating information.

6. Prepare the report of the work

Make a report with all actions according to the tasks from this lab.

Authors: Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A.
Department of SEMIT, NTU «KhPI»
Kharkiv, 2019

REFERENCE LIST

1. Sommerville I. Software engineering / I. Sommerville // Boston: Pearson, 2011. ISBN 0-13-705346-0.
2. IEEE Computer Society. Software Engineering Body of Knowledge (SWEBOK Version 3) [Electronic resource] / Mode of access: www.swebok.org – 12.11.2018.
3. Visual Paradigm Community Edition [Electronic resource] / Mode of access: https://www.visual-paradigm.com/ – 12.11.2018.
4. Richard L. Halterman. Fundamentals of Programming C++ / Richard L. Halterman // Publisher: Southern Adventist University, 2018. – 742 p.
5. Jon Kalb. Title C++ Today: The Beast Is Back / Jon Kalb, Gasper Azman // ISBN-13: 978-1491931660. – Publisher: O'Reilly Media, 2015. – 74 p.

Authors: Melnyk K.V., Borisova N.V., Lutenko I.V., Ershova S.I., Smolin P.A., Grinchenko M.A.
Department of SEMIT, NTU «KhPI»
Kharkiv, 2019